RAVIN: A Resource-aware VNF Placement Scheme with Performance Guarantees

Venkatarami Reddy Chintapalli, Vishal Siva Kumar Giduturi, Bheemarjuna Reddy Tamma, Antony Franklin A
Indian Institute of Technology Hyderabad - INDIA
Email: {cs17resch01007, cs18btech11013, tbr, antony.franklin}@iith.ac.in

NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium | 978-1-6654-7716-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/NOMS56928.2023.10154423

***Abstract*—Network Functions Virtualization (NFV) enables carriers to replace dedicated middleboxes with Virtual Network Functions (VNFs) consolidated on a few shared servers. However, the question of how (and even whether) one can achieve performance related Service Level Objectives (SLOs) with software packet processing in NFV remains open. VNF consolidation causes high variability and unpredictability in throughput and latency of VNFs deployed together. It was shown in our prior work that isolating the processor's Last Level Cache (LLC) and limiting Memory Bandwidth (MB) directly helps in achieving performance isolation among the co-located VNFs. So, in this work, we formulate VNF placement problem with exclusive allocation of LLC and MB resources as a Mixed Integer Linear Program (MILP). Due to its hardness to solve, we also present a heuristic solution named *RAVIN* that enforces performance SLOs for multi-tenant NFV servers while being as much resource-efficient as possible. We demonstrate *RAVIN's* effectiveness in improving resource utilization and in reducing the total number of required servers to deploy VNFs compared to state-of-the-art and baseline approaches.**

***Index Terms*—Network Functions Virtualization (NFV), Virtual Network Function (VNF) Placement, Performance Isolation, Last Level Cache Partitioning, Memory Bandwidth Partitioning, and NFV Orchestrator.**

I. Introduction

In legacy communication networks, Network Functions (NFs) like Deep Packet Inspection (DPI), firewall, etc are typically realized using dedicated network devices, which are commonly known as middleboxes. Although such middleboxes are quite popular and able to handle heavy traffic loads, they are expensive to buy and inflexible to modify or reprogram their functionality to meet ever-changing network requirements. Network Functions Virtualization (NFV) addresses these limitations of (proprietary) middleboxes by separating hardware and software. In NFV, NFs are deployed on commodity servers with software-based implementations known as Virtual Network Functions (VNFs) as it offers ease of operation, cost savings, scalability, and resource sharing among the co-located VNFs [1]. These VNFs are usually deployed in Virtual Machines (VMs) or containers, with one or more dedicated CPU cores of commodity servers. However, the virtualization layer incurs additional overhead that could lead to performance degradation in terms of latency and throughput [2]–[4]. On the other hand, ensuring a minimum Performance Guarantee (PG) is a key requirement for many use cases in communication networks [5]. PG is difficult to achieve before the adoption of virtualization in communication networks, and it is even more challenging now [6].

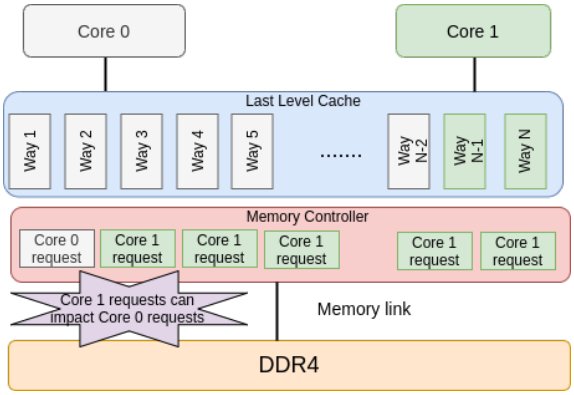

Figure 1: An example of controlling LLC allocations using CAT technology. VNF running in $Core\ 0$ is given more cache ways as it is the high priority one. The other service running in $Core\ 1$ gets fewer cache ways, which results in more frequent cache misses and saturates the memory link.

To alleviate the performance degradation for the deployed network services, commodity servers offer numerous optimization solutions such as Data Plane Development Kit (DPDK) [7], Direct Data I/O (DDIO) technology [8], etc. However, multiple VNFs deployed on the same server may contend (hence interfere with each other) for various shared system resources like CPU, RAM, Last Level Cache (LLC), and Memory Bandwidth (MB), as a result it is difficult to achieve PG [2]. It was shown that the performance interference may cause up to a 50% of throughput degradation when compared to a VNF that runs alone on a commodity server [2], [9]. This performance interference problem has recently gained a lot of attention and research community started exploring resource partitioning mechanisms for ensuring PG to the co-located VNFs in commodity servers [9]–[11].

Most recent works in NFV environment convey that the performance degradation is primarily due to contention for LLC [4], [12], [13] and it can be addressed by using Cache Allocation Technology (CAT) [14]. The CAT provides flexibility to partition the LLC into cache ways and assign them to dedicated cores. However, in a scenario where services with fewer LLC ways experience very frequent cache misses and saturate the memory bandwidth (MB) link, as shown in Fig. 1, this indirectly impacts co-located services with more LLC resources. It means that even though we allocate more cache

ways to some VNFs, they are not benefiting from them, which is overlooked in the literature. Recently, in [11], we reported that it could cause up to a 40% reduction in the achieved throughput. To avoid this, the network operators should also allocate MB efficiently along with LLC ways to achieve PG for the network services deployed on commodity servers in NFV environments.

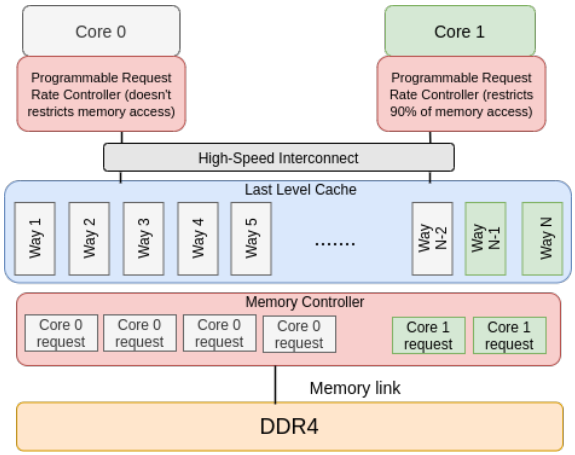

Figure 2: An example of controlling LLC as well as MB allocations. Memory link access for the low priority service running in $Core\ 1$ is restricted by using MBA technology to ensure that the high priority VNF running in $Core\ 0$ indeed gets high priority over the co-located service(s).

Recently Intel introduced Memory Bandwidth Allocation (MBA) technology, which provides indirect and approximate control over MB given to each core. MBA introduces a Programmable Request Rate Controller (PRRC) between cores and shared high-speed interconnect which connects the cores inside each processor. Using PRRC, we can control the amount of MB allocated to different co-located services/VNFs based on their Service Level Objectives (SLOs) (e.g., throughput and latency) and thus improve the performance of high priority VNFs as shown in Fig. 2. In [11], we have shown the importance of jointly allocating LLC and MB in achieving performance isolation through real-time experiments for different VNFs on OpenNetVM platform [15]. But, the existing works ignored the joint management of LLC and MB resources while placing VNF requests in servers. Taking into account only the LLC demand, as followed by *ResQ* [9], causes two main problems. First, the requirement of another resource (i.e., MB) may exceed the system's maximum capacity, leading to performance degradation to all the deployed services on that server. Second, even if we take the maximum limit of MB into consideration along with LLC allocation, it may lead to a significant waste of system resources on the server. In resource-constrained NFV environments, such as edge clouds, the implications of single-dimensional methods on resource efficiency can be more severe.

In this paper, we address the VNF placement problem with the objective of minimizing the total number of servers utilized while guaranteeing performance isolation among the co-located VNFs deployed on each of the servers in NFV environments. VNF profiling results from our recent work [11] are used to map the performance specification in the SLO into an adequate amount of system resources allocation. The main contributions of this work are as follows:

- We formulate VNF placement problem as a Mixed Integer Linear Program (MILP) based optimization problem with the objective of minimizing the total number of servers required to deploy the VNFs while ensuring their PG.
- Owing to the problem's NP-hardness, we propose *RAVIN*, an efficient heuristic scheme for solving VNF placement problem. To characterize the multi-dimensional resource usage states of servers and balance the utilization of both LLC ways and MB, *RAVIN* employs the Balanced Best Fit Decreasing (BBFD).
- Extensive performance results show that *RAVIN* reduces the total number of servers required to deploy VNFs by 9% as compared to state-of-the-art approaches.

The rest of the paper is structured as follows. We provide related works on VNF placement in Section II, and the system model and problem formulation in Section III. We discuss the proposed *RAVIN* scheme in Section IV and present its performance evaluation in Section V. Finally, concluding remarks are given in Section VI.

II. Related Work

In recent years, various packet processing frameworks such as E2 [16], Netbricks [17], and OpenNetVM [15] are developed to address different VNF management issues like traffic steering, load balancing, scalability, etc. However, none of these frameworks studied resource contention problem among co-located VNFs. The authors of [2] investigated the impact of contention for various system resources on performance degradation when multiple types of VNFs are consolidated on a single commodity server. They demonstrated that interference could degrade throughput by up to 50%.

Since the NFV appeared, many works have proposed different techniques to place VNFs on suitable servers. The most involved research on VNF placement can be roughly grouped into two categories: VNF placement with or without consideration of interference impact, as shown in Fig. 3.

VNF placement without considering interference impact: Sang *et al.* [18] reduced VNF placement problem as a classical set cover problem and proposed approximation algorithms. The authors of [19]–[21] studied the availability-aware VNF mapping problem and presented novel algorithms that minimize physical resource consumption. Sun *et al.* [22] designed a flow controller to realize NFV elasticity control and merge under-loaded VNFs for saving energy.

VNF placement by considering interference impact: Recent works on solving performance interference problem can be broadly classified into two categories: predicting performance degradation using machine learning models based on various system-level metrics or partitioning system resources among the co-located services deployed on the commodity

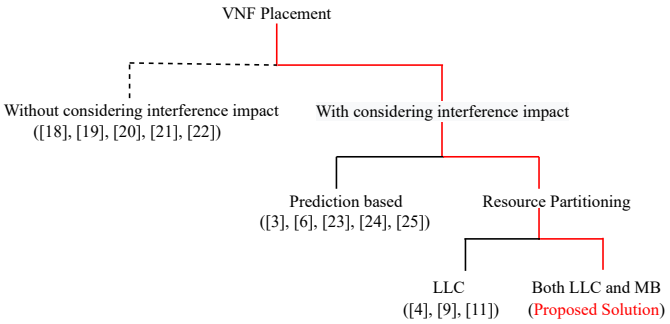

Figure 3: Classification of VNF placement schemes.

server. In the following, we review existing literature under these categories.

1) Predicting performance degradation among co-located services: Recently, some studies have focused on how to place VNFs to avoid severe resource contention. Various VNF placement approaches are proposed in [3], [6], [23]–[25] by considering the impact of interference due to resource contention among co-located VNFs. Among them, [3], [6], [23] developed models that help to predict performance degradation of a VNF when it is co-located with other VNFs, and they also proposed placement solutions using such prediction models. Zhang *et al.* [3] derived an interference metric based on experimental results and then proposed a heuristic approach based on it to solve VNF placement problem which considers interference impact between co-located VNFs. Manousis *et al.* [6] developed a framework named SLOMO for multi-variable performance prediction of VNFs. Madhura *et al.* [23] designed a scalable interference aware clustering algorithm for placement of microservices. All these works focused mainly on predicting the impact of shared resource contention on performance degradation and taking corrective measures like choosing a server with less impact.

2) Using resource partition to address performance degradation: A few works in the literature considered partitioning of the LLC to provide performance isolation [4], [9], [13] among the co-located VNFs. ResQ [9] makes use of CAT to enforce SLOs of co-located VNFs. The authors demonstrated how CAT could be used to achieve throughput and latency guarantees successfully. ResQ employs a two-step offline approach. First, the VNFs must be profiled using a variety of traffic profiles. In the second step, the profiles can be used to improve the placement of the VNFs and the allocation of the LLC ways. However, ResQ does not provide complete isolation among the competing VNFs as they still compete for other system resources like MB. In [11], we have shown the importance of jointly allocating LLC and MB resources to VNFs in achieving performance isolation and then proposed a mechanism for dynamically reallocating these resources to the already deployed VNFs on the server. However, to the best of our knowledge, none of these existing works looked into the joint management of LLC and MB while placing VNFs on the available servers at the disposal of network operators. It is very essential to devise novel approaches to efficiently allocate these scarce system resources among the VNFs while also considering their SLOs and the cost of the servers utilized.

Figure 4: VNF placement procedure: the orchestrator receives a set of VNF requests. It then chooses a server from the cluster of servers with sufficient resources to deploy the VNF.

III. Proposed Work

Network operators and cloud service providers usually deploy VNFs for customers/users to fulfill their SLOs (e.g., throughput and latency) by allocating the required amount of system resources (e.g., CPU, memory, LLC, etc.) as shown in Fig. 4. Here, the NFV orchestrator performs multiple tasks, including VNF placement, VNF scaling, and VNF fault management. In this work, we look into the issues related to VNF placement. The server cluster can accommodate more VNFs with a better VNF placement scheme, resulting in higher revenue for the service provider or lower costs for the operators. As a result, designing of efficient placement schemes that are deployed at the NFV orchestrator has attracted momentous interest.

A. System Model

In this work, we consider limited shareable system resources such as LLC and MB, which are the resources that could cause performance degradation [11] to the co-located VNFs if their allocation is not done efficiently. In comparison to these shareable resources, modern servers have surplus CPU cores and main memory resources and therefore we do not consider them as scarce shared resources in this work. Nonetheless, performance degrades due to increased contentions for LLC and MB [11]. So, it is very much required to efficiently allocate these limited resources to the deployed network services. As in [26], we also assume that the servers in the cluster are homogeneous. It means that all the servers have the same resource configuration with L number of LLC ways and $M\%$ MB. We also assume that the ingress traffic rate for each VNF request is constant or the maximum traffic rate is specified in the request at the time of VNF placement. We need to find minimum number of servers required to deploy R VNFs while ensuring their PG. We make use of VNF profiling to map the performance specification in the SLO into an adequate allocation of system resources. The resource requirements of a VNF are determined by its type and ingress traffic rate, which are retrieved from the profiling lookup table (explained later in this section).

B. VNF Profiling

The main goal of VNF profiling is to efficiently profile each of the VNFs, understand their performance characteristics and build lookup tables, thereby helping the NFV orchestrator to

Table I: Minimum resource combination (x,y) required for different VNFs to achieve the SLO [11]. Here x and y represent number of LLC ways and % of MB required to guarantee the required throughput.

Required Throughput (Gbps)									
VNF	**1-2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Router	(2,10)	(2,10)	(2,10)	(2,20)	(2,30) (3,20)	(2,30) (3,20)	(3,20)	(4,30)	(5,40)
Flow Tracker	(2,10)	(2,10)	(2,10)	(3,50) (4,20)	(5,30)	-	-	-	-
Basic Monitor	(2,10)	(2,10)	(2,10)	(2,20)	(2,20)	(2,20)	(3,30)	(3,30)	(4,40)
Simple Forward	(2,10)	(2,10)	(2,10)	(2,10)	(2,10)	(2,20) (3,10)	(3,30) (2,40)	(3,40)	(4,20)

Table II: Glossary for the Optimization Model

Notation	Description
K	Number of servers used
n	Number of VNF requests
y_j	= 1, if j^{th} server is used; 0, otherwise
x_{ijk}	= 1, if i^{th} VNF request with k^{th} resource configuration is placed in j^{th} server; 0, otherwise
L	Maximum capacity of LLC
M	Maximum capacity of MB
C_i	Number of possible resource configurations for i^{th} VNF request
l_i	LLC requirement for i^{th} VNF request
m_i	MB requirement for i^{th} VNF request

efficiently allocate LLC and MB resources. In [11], we profiled a variety of VNFs available in OpenNetVM platform. Table I shows the lookup table built for these VNFs. The resource required to achieve corresponding throughput is defined as a tuple (l_i, m_i), where l_i and m_i denote the number of LLC ways and the percentage of MB that need to be allocated to the target VNF, respectively. It is worth mentioning that only when sufficient system resources are allocated to the respective VNFs, their achieved throughputs match to their corresponding offered input traffic rates. The combination of bare minimum LLC and MB resources that need to be allocated to achieve the guaranteed performance is distinctive for different VNFs. Moreover, multiple combinations of LLC and MB are possible for a given input traffic rate and the lookup table will return all of them. For example, a VNF may achieve the required throughput when it has been given either 3 LLC ways with 20% MB partitioning or 4 LLC ways with 10% MB partitioning. The orchestrator is responsible to decide which server and which combination of resources to choose from based on the available resources in the cluster of servers and the SLO of the VNF request being processed.

To verify the performance assurance to VNFs by allocating resources based on profiling results, we conducted experiments for multiple scenarios by running a router VNF[1] and *stress-ng* workload (which causes a lot of LLC misses and higher amount of MB usage to induce sufficient stress on the memory system) on a same commodity server. Using *pktgen* tool on a different server, 10 $Gbps$ input traffic is fed to the router VNF. In the first scenario, we deploy router VNF alone on the server and observe its maximum achievable performance in an isolated setting. It is considered as the benchmarking result. Next, we deploy router together with *stress-ng* without any resource reservation (unmanaged) to either of them on the same server. Since we do not reserve the resources, the performance of the router degrades because *stress-ng* consumes more system resources, therefore causing performance interference to its co-located services i.e., router VNF. Finally, router VNF has been configured with five LLC ways and 40% of MB (corresponding to 10 $Gbps$ ingress traffic rate for router as shown in Table I). It achieves similar performance as that in the first scenario i.e., as if the VNF is running alone in the system. Fig. 5 depicts the performance of router VNF in these three scenarios. For each scenario, 10 experiments are performed and the results are plotted with 95% confidence intervals. It is clear that by jointly allocating LLC and MB, it is feasible to offer PG for a target VNF regardless of other services deployed on the same server. It is to be noted that the profiling approach we employed is generic and could be applied to any other type of VNF.

[1]https://github.com/sdnfv/openNetVM/tree/master/examples

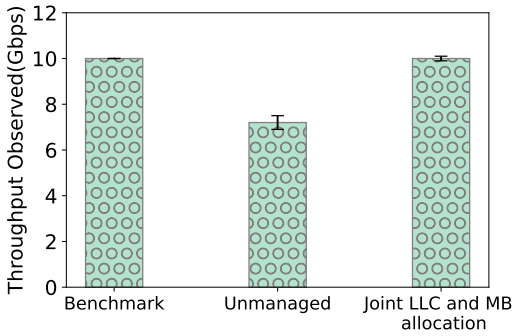

Figure 5: Throughput of router VNF in various experimental scenarios.

C. Problem Formulation

In this section, we mathematically formulate VNF placement problem as an MILP with the objective of minimizing the total number of servers required in the cluster for deploying a given set of VNF requests. Notations used are given in Table II.

$$Minimize: \quad K = \sum_{j=1}^{n} y_j \tag{1}$$

subject to the following constraints:

$$\sum_{i \in I}^{n} \sum_{k=1}^{C_i} l_i x_{ijk} \leq L y_j \quad \forall j \in [1, n] \tag{2}$$

$$\sum_{i \in I}^{n} \sum_{k=1}^{C_i} m_i x_{ijk} \leq M y_j \quad \forall j \in [1, n] \tag{3}$$

$$\sum_{j=1}^{n}\sum_{k=1}^{C_i} x_{ijk} = 1 \quad \forall\ i \in I \tag{4}$$

$$y_j \in \{0,1\} \quad \forall j \in [1,n] \tag{5}$$

$$x_{ijk} \in \{0,1\} \quad \forall\ i \in I,\ \forall j \in [1,n],\ \forall k \in [1,C_i] \tag{6}$$

$$l_i \in Z^+,\ m_i \in Z^+ \tag{7}$$

Eq. (1) shows that the objective of the MILP model is to minimize the total number of servers needed to accommodate n VNF requests. Eq. (2) ensures that the sum of allocated LLC resources in a server does not exceed the total LLC capacity of the server. Eq. (3) ensures that the sum of allocated MB resources in a server does not exceed the total MB capacity of the server. Eq. (4) ensures that for each VNF i at most one resource combination (from the lookup table) is selected.

Since the problem of optimal VNF instance placement has been proved to be NP-hard [21], [27], we propose an efficient heuristic mechanism named *RAVIN* in this work. *RAVIN* tries to efficiently utilize both the LLC and MB resources while minimizing the total number of servers required for deploying the VNF requests.

IV. RAVIN: PROPOSED HEURISTIC MECHANISM FOR VNF PLACEMENT WITH PERFORMANCE GUARANTEES

In this section, first we describe a placement strategy used for balancing the usage of multiple resources. We then present RAVIN which employs this placement strategy while processing VNF requests.

A. Placement Strategy

Efficient utilization of system resources helps in reducing the total number of servers powered-on in the cluster. Mechanisms which consider a single resource while placing VNFs cannot be directly applied in the case of multiple resources. Since these mechanisms optimize one resource it may cause under-utilization of other resources which in turn can lead to the usage of more servers in the cluster. So, we design a placement strategy to optimize the utilization of multiple resources and therefore reduce the number of servers required. This problem is a classical extension of vector bin packing problem which is NP-hard [28]. It is well known that Best Fit Decreasing (BFD) is an efficient algorithm for the linear dimensional bin packing problem. In BFD, objects are arranged in decreasing order and an object is allocated in a bin with least amount of space left after the allocation and this process is repeated until all the objects are allocated in bins. In this work, we generalize BFD for multiple dimensions by consolidating multiple resource constraints into a single constraint and then applying our heuristic to this single consolidated constraint to optimize the utilization of multiple resources.

To avoid either type of resource becoming the bottleneck while placing the VNFs on the servers, we balance the resource utilization of both the resources in each server by employing Balanced Best Fit Decreasing (BBFD). The BBFD takes both resources into account and tries to balance resource utilization on the server. Specifically, the BBFD minimizes the variance of the allocated resources in each server using the following equation.

$$Var(u^{(n)}) = \sum_{k}(u_k^{(n)} - \overline{u}^{(n)})^2 \tag{8}$$

where $u_k^{(n)}$ represents the utilization of type-k resource of server $s^{(n)}$, $\overline{u}^{(n)}$ represents the mean value of $u_1^{(n)}, u_2^{(n)}, \ldots, u_k^{(n)}$, and $u^{(n)} = [u_1^{(n)}, u_2^{(n)}, \ldots, u_k^{(n)}]$. Accordingly, when selecting an object (VNF request) for a bin (server), the BBFD tries to place the object that can balance the resources of a given bin (server). In the following, we present *RAVIN* that uses the BBFD approach to place VNFs efficiently.

Algorithm 1: RAVIN

```
1  Input: List of VNF requests and their resource requirements
2  Output: Number of servers required
3  begin
4      Create two lists such as LLClist and MBlist based on
         the resource configurations available for VNFs
5      Bin_L ← BBFD(LLClist)
6      Bin_M ← BBFD(MBlist)
7      if |Bin_L| < |Bin_M| then
8          Deploy according to Bin_L configuration
9          Return |Bin_L|
10     end
11     else
12         Deploy according to Bin_M configuration
13         Return |Bin_M|
14     end
15 end
16 begin
17     Procedure BBFD(V)
18     Initialize V_a ← Φ, V_u ← V, s={}
19     while V_u ≠ Φ do
20         Launch a new server s_new
21         while s_new has enough resources for v_k ∈ V_u do
22             j ← arg min_{v_k ∈ V_u} Var(u^(n))
23             Place VNF v_j on s_new
24             Update available resources in s_new
25         end
           s ← s ∪ s_new
           V_a ← VNFs in s_new
           // Remove all the VNFs in V_a from V_u
           V_u ←V_u \ V_a
26     end
       Return s
27 end
```

B. Proposed Heuristic Algorithm: RAVIN

The working of *RAVIN* is given in Algorithm 1. Notations used are given in Table III. It takes a set of VNF requests and their resource requirements (from the lookup table based on their required throughput requirements) as the inputs and returns the number of servers required to deploy these VNFs as the output. It is seen that multiple resource combinations of LLC and MB may satisfy throughput requirements for a given VNF. Line 4 creates two lists based on the resource combinations available for VNFs namely $LLClist$ and $MBlist$. $LLClist$ picks the resource combination greedily on LLC (i.e., the one with least

Table III: Notations

Notation	Description
V_a	A set containing allocated VNF requests
V_u	A set containing unallocated VNF requests
v_k	k^{th} VNF request
$Var(u^{(n)})$	Variance of all the remaining resources in the server n
v_j	j^{th} VNF request for which variance is minimum
s	A set containing all of the accommodated servers

LLC ways) from the available entries for each VNF. Similarly, for $MBlist$, which is greedy on MB. *RAVIN* uses the BBFD procedure for these two lists and determines which list requires least number of servers; accordingly, VNFs are deployed on those identified servers. The procedure for BBFD is explained as follows. For each VNF, it picks a server and calculates the variance using Eq. (8) with all the requests separately which are not yet allocated. The request which gives the least variance is then placed in the server by allocating it with the required resources. This process is repeated until the server cannot accommodate any further requests or until all requests have been satisfied. In case the server is fully packed and more requests are left, then a new server is picked, and the same process is repeated until all the requests are accommodated.

C. Complexity Analysis

Let N represent the number of VNF requests and M represent the maximum number of resource combinations for any VNF. In Algorithm 1, we first generate two lists by greedily selecting resource combinations on either LLC or MB for each VNF. Thus, the time complexity for creating these lists is $\mathcal{O}(N \times M)$. Then *RAVIN* calls the BBFD procedure to place VNFs by balancing resource utilization. BBFD picks a new server and iteratively chooses the VNF requests from V_u that balance server resource utilization by using Eq. 8 and allocates them to that server. This process is repeated until the server cannot accommodate any more VNF requests due to resource crunch. This process takes $\mathcal{O}(N^2)$ time. Thus, the time complexity of *RAVIN* is $\mathcal{O}((N \times M) + N^2)$.

V. Performance Evaluation

We have evaluated the proposed *RAVIN* scheme in terms of the total number of servers required to deploy a given number of VNF requests and compared it against state-of-art and baseline schemes. All the schemes are implemented in C++ and IBM CPLEX tool is used to solve the proposed MILP formulation. VNF requests are generated randomly from the profiled VNFs listed in Table I, and each VNF's ingress traffic rate is chosen randomly from the range of $1 - 10$ $Gbps$. Homogeneous servers are considered with the configuration of 11 LLC cache ways and 100% MB in the cluster for placing the VNF requests. It is worth noting that for each combination of the VNF requests, we executed 100 different runs and average values are plotted.

A. Comparison Schemes

In addition to *RAVIN*, we have implemented the following schemes for comparison.

- ***ResQ [9]:*** Depending upon the requested traffic rate of a target VNF, this scheme allocates minimum number of LLC ways to the target VNF without considering its MB requirements. It means *ResQ* follows a simple first-fit bin packing heuristic using Intel's CAT mechanism.
- ***NOMS* [27]:** This scheme, proposed in [27] (referred to as *NOMS* in this work), considers two resources such as CPU and memory while placing VNFs and uses a metric s^i which is the ratio of these resources. A ratio $s^i = d_1^i/d_2^i$ is computed for i^{th} VNF request where d_1^i is the resource requirement for the first resource and d_2^i is the requirement for the second resource. A similar ratio $s = r_1/r_2$ is computed for each server where r_1 is the available resources of the first kind and r_2 is the available resources of the second kind. *NOMS* takes the absolute difference of these metrics and places the VNF on the server if the result of the obtained metric is the minimum of all the requests. We use the same mechanism for LLC and MB resources in place of CPU and memory in this work and observe its performance for a fair comparison.
- ***LLC-greedy (ResQ with MB consideration):*** This scheme allocates resource combination having the least number of LLC ways to the target VNF in correspondence to its input traffic rate. Unlike *ResQ*, this scheme allocates both LLC ways and MB to the target VNF.
- ***MB-greedy:*** In this scheme, a resource combination entry that takes the least amount of MB gets allocated to the target VNF. Along with MB, this scheme also allocates a certain number of LLC ways.

The VNF placement scheme, *MB-greedy* is the baseline scheme, while *ResQ* and *NOMS* are the state-of-the-art schemes. We also consider *LLC-greedy* which is a variation of *ResQ* for performance comparison with *RAVIN* scheme.

B. Performance Metrics

We vary the number of VNF requests for placement on servers in the cluster and measure the following performance metrics:

- **Number of servers used:** This metric represents the total number of servers required to deploy a given set of VNF requests.
- **Average resource utilization ratio:** Resource utilization of a server is defined as the ratio of number of resources consumed (i.e., LLC and MB) in the server to the total number of resources present in that server. It corresponds to the percentage of the resources that have been allocated for the VNF requests. In this metric, we report the average resource utilization of all servers that got launched for serving the VNF requests.

C. Performance Results

1) Effect of no. of VNF requests on no. of servers used: In this experiment, we measure the total number of servers needed by varying the number of VNF requests from 100 to 1500 in steps of 200. We have considered *LLC-greedy*, *MB-greedy*, and *NOMS* schemes for comparison with *RAVIN*.

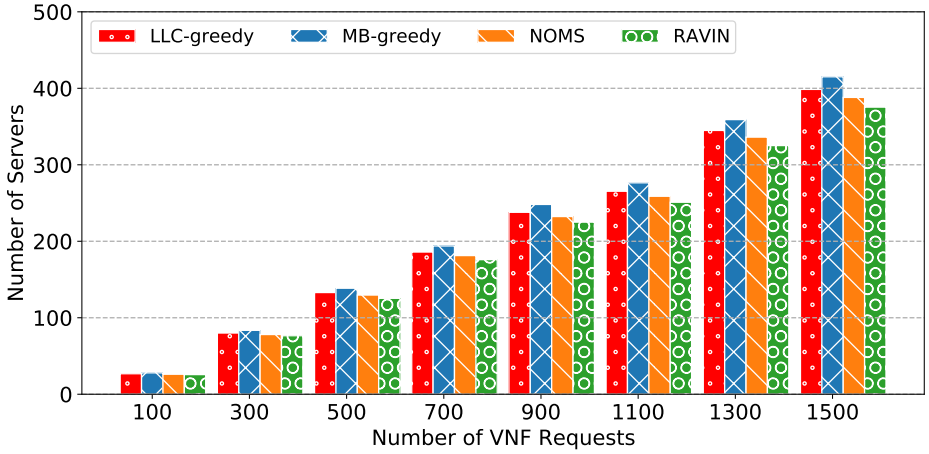

Figure 6: Number of servers needed vs. VNF requests for different VNF placement schemes.

ResQ is not considered here because it allocates LLC ways to VNFs greedily without considering MB which leads to SLO violations to the deployed VNFs. This is explained later in this section. Fig. 6 shows the variation in the number of servers launched for different schemes w.r.t. number of VNF requests. The figure shows that *LLC-greedy*, *MB-greedy*, and *NOMS* result in a higher number of powered-on (active) servers than *RAVIN*. We also see a gradual increase in the number of servers in all schemes as the number of requests increases. Greedy schemes such as *LLC-greedy* and *MB-greedy* require 9% and 10% more servers than *RAVIN*, respectively. This performance benefit in *RAVIN* is due to the efficient allocation of both LLC and MB resources rather than solely focusing on a single type of resource. In addition, we observe approximately 4% increase in performance by using *RAVIN* over *NOMS* scheme. Even though *NOMS* considers both the resources, lack of resource balancing results in a higher number of servers than that of *RAVIN*. *RAVIN's* improvement is considered significant given that 1% difference in the number of required servers corresponds to a savings of more than ten servers in the cluster.

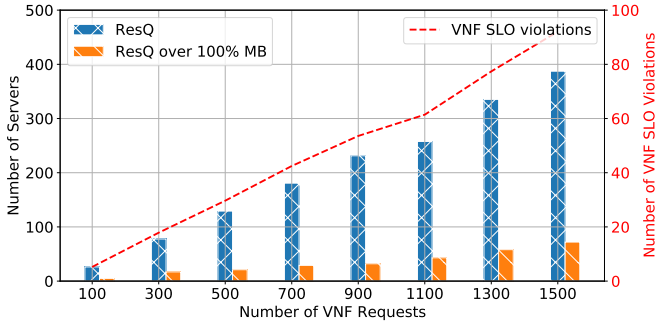

Figure 7: Number of servers in which VNFs are deployed vs. number of VNF SLO violations by ResQ scheme.

ResQ scheme allocates LLC ways to VNFs greedily without considering MB allocation. In doing so, the total MB requirements of all the VNFs to satisfy their SLOs exceeded the maximum limit of the system i.e., 100%. If a VNF does not get the required resources, we consider the SLO of that VNF is violated. To calculate the number of SLO violations incurred for *ResQ*, we removed the VNFs one by one until the total MB requirements of the rest of the VNFs do not exceed the maximum limit, such that the number of SLO violations is reduced to the extent possible. Fig. 7 shows a combined plot of the number of servers launched and the number of servers that exceeded 100% MB utilization by *ResQ* scheme (Y1-axis), and the number of VNFs that experienced SLO violations (Y2-axis). *LLC-greedy* scheme is an extension of *ResQ* that takes MB into consideration as well. As seen in the figure, *ResQ* is causing SLO violations to some of the deployed VNFs, so we did not present the plots of *ResQ* separately in this work.

2) Effect of number of VNF requests on resource utilization ratio:

We evaluated how balanced is the resource utilization across different resources in terms of the resource utilization ratio. Figs. 8 and 9 show variations in average resource utilization of LLC and MB of different schemes, respectively. It is observed that *LLC-greedy* and *MB-greedy* schemes allocate resources greedily by picking one of the resources, leading to increased consumption of the other resource. Whereas *RAVIN* makes use of both resources in a balanced way to ensure better utilization of both resources. We can see that the resource utilization of LLC and MB of *RAVIN* is approximately 94% and 83%, respectively. In case of *LLC-greedy*, the LLC resource utilization is 90% and the corresponding MB utilization is 79%. On the other hand, for *MB-greedy*, the resource utilization of LLC is 96% and corresponding MB utilization is only 66%. It is clear that these greedy schemes assign resource combinations by acting greedily to conserve one resource while depleting another. *RAVIN* makes a prominent improvement in resource utilization among all the schemes because it uses both the resources in a balanced manner which leads to the usage of only fewer servers. Even compared to *NOMS*, the resource utilization of our proposed *RAVIN* scheme is better.

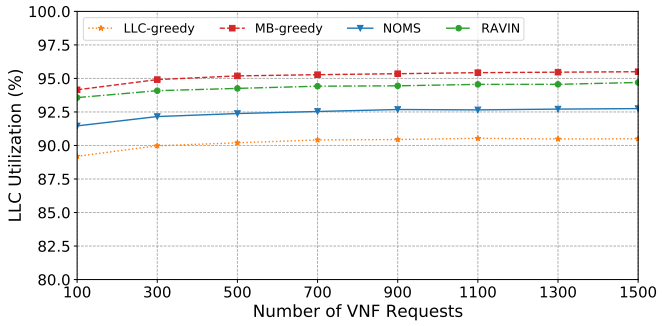

Figure 8: LLC utilization of different schemes.

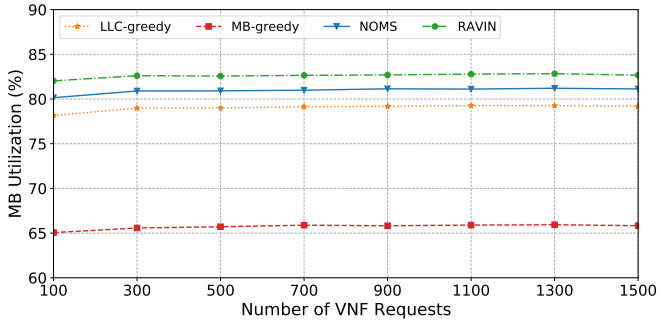

Figure 9: MB utilization of different schemes.

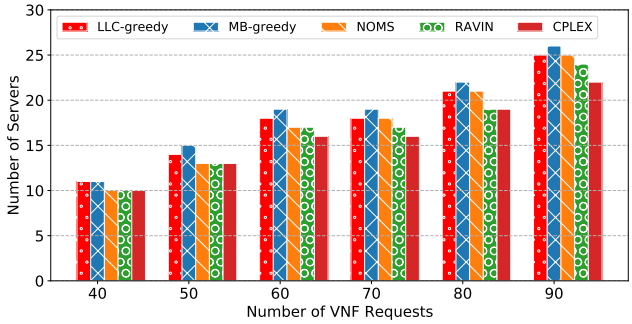

Figure 10: Number of servers comparison with different schemes and optimal solution using CPLEX.

D. Optimization Results

We verified how close the performance of *RAVIN* w.r.t. MILP model is by considering a smaller setup. Here, we varied the number of VNF requests from 30 to 90 and observed the number of servers required by all the schemes (refer Fig. 10). It is clearly evident that *RAVIN* is very close (less than 5% in all the cases) to the optimal. *NOMS* is also behaving similar to *RAVIN* with a difference in the performance of 8% to the optimal. Whereas other schemes have taken more servers compared to *RAVIN*, and a similar trend has been observed even when we tested these schemes by increasing the number of VNF requests. We have excluded *ResQ* in this study as it does not offer performance isolation and causes SLO violations to the VNFs deployed. In contrast, *RAVIN* ensures performance isolation with near-optimal performance.

E. Discussion

The proposed *RAVIN* scheme relies on profiling to characterize VNFs. However, we can minimize the number of experiments required for profiling each VNF by adopting the methodology given in [29]. We argue that the task of profiling is not overhead as it takes place only once for each VNF and can be done offline. Moreover, we target services that require performance isolation, which is a very essential requirement for VNFs. Profiling the resource requirements of various VNFs to achieve performance isolation among co-located VNFs has many potential applications. In this section, we provide some directions on how to use the observations and results shown in this work to address the VNF placement problem by considering Non-Uniform Memory Access (NUMA) impact on servers and exploring VNF migration possibilities to maximize the number of VNFs with performance guarantees.

NUMA-awareness: Recently, network and cloud service providers started deploying NUMA based servers to accommodate more VNF requests. CPU cores are grouped into NUMA nodes in these servers, resulting in performance bottlenecks due to cross-node memory access and intra-node resource contention [30]–[32]. Choosing which CPU core to use in VNF placement is also important for ensuring performance guarantees.

VNF migration: The input traffic rate of VNF changes dynamically. As the input traffic rate to the VNF changes over time, the required system resources to that VNF also needs to be reconfigured rapidly to meet the required performance guarantees. With dynamic allocation of resources, we can ensure performance isolation for even more number of VNFs. It would be an interesting study to extend this work by providing migration mechanisms when resources are insufficient based on the dynamic traffic rate of VNFs i.e., VNFs can be migrated to another NUMA node on the same server or to a different server completely.

VI. Conclusions and Future Directions

Ensuring performance guarantees in NFV environments is challenging when multiple VNFs share system resources, especially LLC and MB. We formulated the VNF placement problem as an MILP, with the aim of minimizing the number of servers required to deploy the VNFs. Due to NP-hardness of the problem, we also presented a heuristic solution named *RAVIN*, which balances resource utilization while ensuring performance isolation among the co-located VNFs on a server. The results of the performance evaluation demonstrated the effectiveness of *RAVIN* in utilizing resources compared to other baseline approaches. Through simulations, we demonstrated that *RAVIN* could significantly reduce the number of servers required for deploying VNF requests by 9% compared to state-of-the-art schemes. Future directions include extending the proposed solution to address the placement and resource allocation of a chain of VNFs. In addition, we would like to design ML-based solutions for dynamically allocating system resources like LLC and MB to the VNFs deployed.

REFERENCES

[1] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys Tutorials*, 18(1):236–262, 2016.

[2] C. Zeng, F. Liu, S. Chen, W. Jiang, and M. Li. Demystifying the performance interference of co-located virtual network functions. In *Proc. of IEEE INFOCOM*, 2018.

[3] Qixia Zhang, Fangming Liu, and Chaobing Zeng. Online adaptive interference-aware vnf deployment and migration for 5g network slice. *IEEE/ACM Transactions on Networking*, 29(5):2115–2128, 2021.

[4] Paul Veitch, Edel Curley, and Tomasz Kantecki. Performance evaluation of cache allocation technology for nfv noisy neighbor mitigation. In *Proc. of IEEE NetSoft*, 2017.

[5] Shuaishuai Guo, Binbin Lu, Miaowen Wen, Shuping Dang, and Nasir Saeed. Customized 5g and beyond private networks with integrated urllc, embb, mmtc, and positioning for industrial verticals. *IEEE Communications Standards Magazine*, 6(1):52–57, 2022.

[6] Antonis Manousis, Rahul Anand Sharma, Vyas Sekar, and Justine Sherry. Contention-aware performance prediction for virtualized network functions. In *Proc. of ACM SIGCOMM*, 2020.

[7] Intel. DPDK: Data Plane Development Kit: Programmer's Guide. Intel Corporation, Feb 2017.

[8] Intel. DDIO: Intel Data Direct I/O Technology Overview. Intel White Paper, 2012.

[9] Amin Tootoonchian, Aurojit Panda, Chang Lan, Melvin Walls, Katerina Argyraki, Sylvia Ratnasamy, and Scott Shenker. Resq: Enabling slos in network function virtualization. In *Proc. of ACM NSDI*, 2018.

[10] Haoran Qiu, Subho S Banerjee, Saurabh Jha, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. {FIRM}: An intelligent fine-grained resource management framework for {SLO-Oriented} microservices. In *Proc. of ACM OSDI*, 2020.

[11] Venkatarami Reddy Chintapalli, Madhura Adeppady, Bheemarjuna Reddy Tamma, et al. Restrain: A dynamic and cost-efficient resource management scheme for addressing performance interference in nfv-based systems. *Journal of Network and Computer Applications*, page 103312, 2022.

[12] Mihai Dobrescu, Katerina Argyraki, and Sylvia Ratnasamy. Toward predictable performance in software {Packet-Processing} platforms. In *Proc. of ACM*, 2012.

[13] Bin Li, Yipeng Wang, Ren Wang, Charlie Tai, Ravi Iyer, Zhu Zhou, Andrew Herdrich, Tong Zhang, Ameer Haj-Ali, Ion Stoica, et al. Rldrm: closed loop dynamic cache allocation with deep reinforcement learning for network function virtualization. In *Proc. of IEEE NetSoft*, 2020.

[14] Intel. Introduction to Cache Allocation Technology in the Intel Xeon Processor E5 v4 Family. Intel Corporation, 2016.

[15] Wei Zhang, Guyue Liu, Wenhui Zhang, Neel Shah, Phillip Lopreiato, Gregoire Todeschi, K.K. Ramakrishnan, and Timothy Wood. Opennetvm: A platform for high performance network service chains. In *Proc. of ACM HotMiddlebox*, page 26–31, 2016.

[16] Shoumik Palkar, Chang Lan, Sangjin Han, Keon Jang, Aurojit Panda, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. E2: A framework for nfv applications. In *Proc. of ACM SOSP*, 2015.

[17] Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. Netbricks: Taking the v out of nfv. In *Proc. of ACM OSDI*, 2016.

[18] Yu Sang, Bo Ji, Gagan R Gupta, Xiaojiang Du, and Lin Ye. Provably efficient algorithms for joint placement and allocation of virtual network functions. In *Proc. of IEEE INFOCOM*, 2017.

[19] Jingyuan Fan, Chaowen Guan, Yangming Zhao, and Chunming Qiao. Availability-aware mapping of service function chains. In *Proc. of IEEE INFOCOM*, 2017.

[20] Jiao Zhang, Zenan Wang, Chunyi Peng, Linquan Zhang, Tao Huang, and Yunjie Liu. Raba: Resource-aware backup allocation for a chain of virtual network functions. In *Proc. of IEEE INFOCOM*, 2019.

[21] Chenxi Qiu, Haiying Shen, and Liuhua Chen. Towards green cloud computing: Demand allocation and pricing policies for cloud service brokerage. *IEEE Transactions on Big Data*, 5(2):238–251, 2018.

[22] Chen Sun, Jun Bi, Zili Meng, Tong Yang, Xiao Zhang, and Hongxin Hu. Enabling nfv elasticity control with optimized flow migration. *IEEE Journal on Selected Areas in Communications*, 36(10):2288–2303, 2018.

[23] Madhura Adeppady, Carla Fabiana Chiasserini, Holger Karl, and Paolo Giaccone. iplace: An interference-aware clustering algorithm for microservice placement. In *Proc. of IEEE ICC*, 2022.

[24] Marco Savi, Massimo Tornatore, and Giacomo Verticale. Impact of processing-resource sharing on the placement of chained virtual network functions. *IEEE Transactions on Cloud Computing*, 9(4):1479–1492, 2021.

[25] Yanyan Mu, Lei Wang, and Jin Zhao. Energy-efficient and interference-aware vnf placement with deep reinforcement learning. In *Proc. of IEEE IFIP Networking*, pages 1–9, 2021.

[26] Jian Zhao, Hongxing Li, Chuan Wu, Zongpeng Li, Zhizhong Zhang, and Francis CM Lau. Dynamic pricing and profit maximization for the cloud with geo-distributed data centers. In *Proc. of IEEE INFOCOM*, 2014.

[27] Angelos Pentelas, George Papathanail, Ioakeim Fotoglou, and Panagiotis Papadimitriou. Network service embedding with multiple resource dimensions. In *Proc. of IEEE/IFIP NOMS*, 2020.

[28] EG Co man Jr, MR Garey, and DS Johnson. Approximation algorithms for bin packing: A survey. *Approximation algorithms for NP-hard problems*, pages 46–93, 1996.

[29] M. G Khan et al. A performance modelling approach for sla-aware resource recommendation in cloud native network functions. In *Proc. of IEEE NetSoft*, 2020.

[30] Heng Yu, Zhilong Zheng, Junxian Shen, Congcong Miao, Chen Sun, Hongxin Hu, Jun Bi, Jianping Wu, and Jilong Wang. Octans: Optimal placement of service function chains in many-core systems. *IEEE Transactions on Parallel and Distributed Systems*, 32(9):2202–2215, 2021.

[31] George Papathanail, Angelos Pentelas, and Panagiotis Papadimitriou. Towards fine-grained resource allocation in nfv infrastructures. In *Proc. of IEEE GLOBECOM*, 2021.

[32] Venkatarami Reddy Chintapalli, Sai Balaram Korrapati, Bheemarjuna Reddy Tamma, and Antony Franklin A. Numasfp: Numa-aware dynamic service function chain placement in multi-core servers. In *Proc. of IEEE/ACM COMSNETS*, 2022.